PATENT COOPERATION TREATY

PCT

NOTIFICATION OF ELECTION

(PCT Rule 61.2)

From the INTERNATIONAL BUREAU

To:

Assistant Commissioner for Patents
United States Patent and Trademark Office
Box PCT
Washington, D.C.20231
ETATS-UNIS D'AMERIQUE

in its capacity as elected Office

Date of mailing (day/month/year) 23 August 2000 (23.08.00)	
International application No. PCT/SG98/00102	**Applicant's or agent's file reference** FP1103
International filing date (day/month/year) 16 December 1998 (16.12.98)	**Priority date** (day/month/year)
Applicant NGAIR, Teow, Hin et al	

1. The designated Office is hereby notified of its election made:

   [X] in the demand filed with the International Preliminary Examining Authority on:

   30 June 2000 (30.06.00)

   [ ] in a notice effecting later election filed with the International Bureau on:

2. The election [X] was [ ] was not

   made before the expiration of 19 months from the priority date or, where Rule 32 applies, within the time limit under Rule 32.2(b).

The International Bureau of WIPO **34, chemin des Colombettes** **1211 Geneva 20, Switzerland** Facsimile No.: (41-22) 740.14.35	Authorized officer Pascal Piriou Telephone No.: (41-22) 338.83.38

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 9/46, 9/54

A1

(11) International Publication Number: WO 00/36506

(43) International Publication Date: 22 June 2000 (22.06.00)

(21) International Application Number: PCT/SG98/00102

(22) International Filing Date: 16 December 1998 (16.12.98)

(71) Applicant (*for all designated States except US*): KENT RIDGE DIGITAL LABS [SG/SG]; 21 Heng Mui Keng Terrace, Singapore 119613 (SG).

(72) Inventors; and
(75) Inventors/Applicants (*for US only*): NGAIR, Teow, Hin [SG/SG]; 334 Kang Ching Road #13-254, Singapore 610334 (SG). PANG, Hwee, Hwa [SG/SG]; 19 Shelford Road #01-42, Singapore 288408 (SG).

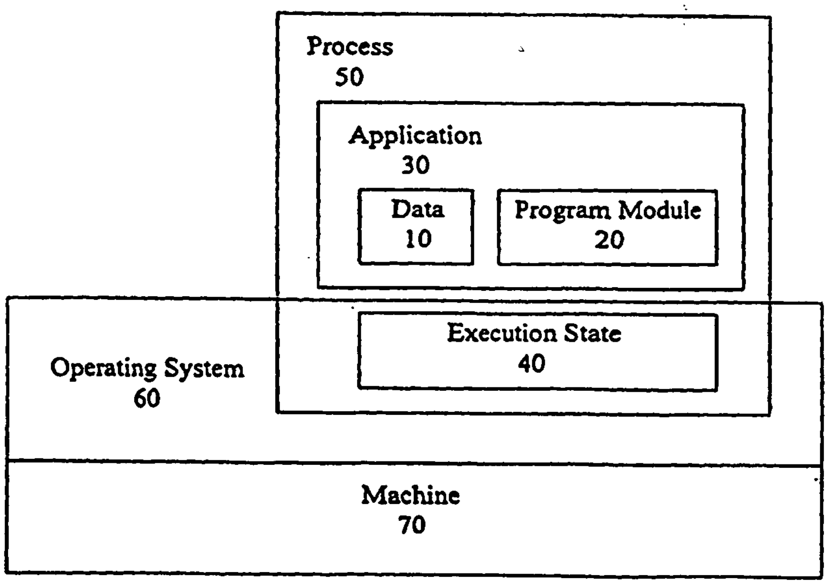
(74) Agent: GREENE-KELLY, James, Patrick; Lloyd Wise, Tanjong Pagar, P.O. Box 636, Singapore 910816 (SG).

(81) Designated States: JP, SG, US, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published

With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: PROCESS ORIENTED COMPUTING ENVIRONMENT



(57) Abstract

A novel software paradigm is disclosed in which processes comprising data, program modules and execution states are treated as first-class entities that may be operated on directly. By forming a hibernaculum that "stores" such a process, the process may be enabled to migrate, or be incorporated into another process, or incorporate another process within it, or mutate by either dropping or loading selected elements. The paradigm provides a computing environment in which process migration and evolution may be facilitated.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

PROCESS ORIENTED COMPUTING ENVIRONMENT

This invention relates to a computing environment, and in particular to a process oriented computing environment, and also to a method for controlling the migration and evolution of processes within the environment.

Recent years have seen a number of developments in computing science regarding how elements within a software application are treated and handled. In this context the most basic elements to be found within a software application are data and program modules. Traditional procedural programming paradigms focus on the logic of the software application, so a program is structured based on program modules. One uses the program modules by explicitly supplying to them the data on which they should operate. A potential pitfall of this paradigm is that it is difficult to guarantee type safety, ie that the data being passed to a program module is of the correct type.

More recently there has been a move towards an object-oriented paradigm. In this paradigm, programs are structured around objects, with each object representing an entity in the world being modeled by the software application. Each object manages its own data (state), which are hidden from the external world (other objects and programs). An object in a program interacts with another object by sending it a message to invoke one of its exposed program modules (method). This paradigm imposes better control and protection over internal data, and helps to structure complex applications designed and implemented by a team of programmers. An example of an object-oriented environment can be found in US 5,603,031. This discloses an environment in which new agents (essentially objects) consisting of data and program modules can be sent between machines.

While object-oriented paradigm represents a significant advance in software engineering, the data and modules that constitute each object are static. The paradigm is still inadequate for writing programs that must evolve during execution, eg programs that need to pick up, drop, or substitute selected modules. There have been several attempts at overcoming this limitation. For example, work described in US Patents 4954941, 5175828, 5339430 and 5659751 address techniques for re-linking or re-binding selected software modules dynamically during runtime. Also Microsoft's Win32 provides for

explicit mapping and un-mapping of dynamic linked libraries into the address space of a process through the LoadLibrary and FreeLibrary calls. With this prior art, however, the prototype or specification of functions and symbols are fixed beforehand and compiled into application programs. This means that an object cannot invoke a module of another object for which the specification is not known at compile time.

Another shortcoming of both traditional procedural and object-oriented paradigms is that programs consist only of data and program modules, but not the transient information that capture the entire state of execution during runtime. This makes it difficult to interrupt a running program and to migrate it to another machine. Generally it is only possible to transfer the saved data file of completed applications. As a very simple example of this problem, while a word processing document file or a spreadsheet file may be transferred from one machine to another, it is not possible to do so while the file is currently being worked on without reverting to a saved version. Transient information is lost. In the case of a word processing file, for example, this means that any "undo editing" function cannot be used by the receiving machine on the file it has just received because the transient "undo" information is not part of the data file.

At present, such migrations have been effected from outside the program, as utilities in the computing environment. Examples include Amoeba, Charlotte, Sprite and Condor. Such utilities work by taking a snapshot of the running program (or core dump), and by resuming execution on the recipient machine from the snapshot. Unfortunately the migration utilities have no knowledge of the usage requirements and semantics of the components of the running program, and so there is no way to adapt it to the new computing environment. Consequently, the migrated program most likely cannot run in a different computing environment from the original one, such as when the machines have different devices like displays, hard disks and sound cards. Even in cases where it does, it would not run with the same level of efficiency, for example because the machines have different amounts of main memory.

In this specification the following terms will be used with the following meaning:

"First class entity": an object that can be manipulated directly.

"Process": a combination of data, program module(s) and current execution state.

"Execution state": the values and contents of transient parameters such as the contents of registers, frames, counters, look-up tables and the like.

It is an object of the present invention to provide a computing environment that treats data, program modules and processes all as first class entities and which facilitates the migration and evolution of a process during runtime.

According to the present invention there is provided a computing environment wherein a process comprising a set of data and/or program modules and execution state is treated as an entity that can be transferred between components of the environment, and wherein a said process can be subject to evolutionary operations.

By means of this arrangement a process itself is treated as a first class entity in addition to the traditional first class entities of data and program modules, and the first class entities of object-oriented programming that combine data and program modules. Thus a process can be transferred between machines together with all transient runtime data that comprises the current execution state. This allows an application to be transferred between machines during runtime and to continue running without any interruptions. In addition, however, not only can a process migrate but it can also be subject to evolutionary operations and can thus evolve, either within a first machine or during migration. In addition not only can a process be transferred between different hardware components directly, but a process may be transferred to a memory device for storage preceding subsequent transfer to another hardware component or even back to the first component. Furthermore processes or sub-processes forming a subset of a process may be transferred between processes themselves, and such processes may be running on different hardware components or may be all running within a single hardware component. It is important to appreciate therefore that in the present invention the components of the environment between which a process may be transferred may comprise hardware and/or software elements of the computing environment.

In a preferred embodiment of the invention a construct is formed comprising a set of data and/or program modules and execution state of a first process, and wherein the evolutionary operations are performed by functions operating on a said construct. In particular the construct is formed by a construct operation that suspends all active threads of the first process and creates a new process comprising at least some of the data and/or

program modules and execution state of the first process, and stores the new process in a data area of the said process. While the construct may comprise all of the first process, more commonly the construct will comprise only data, program modules and execution state falling within lists that are passed to the construct operation. Preferably the construct may also be formed with an authorising signature in order to confirm its authenticity if it is to be transferred to another component of the computing environment.

Simple evolutionary operations can be performed on the process stored in the construct without it being transferred to any other component. In particular the process can be modified by either selective deletion of objects from within the process, and/or by selective loading or reloading of object into the process.

In more sophisticated embodiments of the invention, however, a process may be subject to evolutionary operations that include the incorporation into a first process of objects from a second process. Again this may be achieved by firstly forming a construct comprising at least some of the data and/or program modules and execution state from the second process and then transferring this construct to the first process. Preferably the construct is formed by a construct operation that suspends all active threads of said second process and creates a new process comprising some of the data and/or program modules and execution state of said second process, and stores said new process in a data area of said second process. Again while the construct may comprise all of the second process, it may also comprise only data, program modules and execution state falling within lists that are passed to said construct operation. An authorising signature may also be provided to the construct.

There are, however, a number of ways that this can be done and in which the elements of the two processes can be combined.

In a simple embodiment after the construct is transferred the second process stored within the construct is caused to be activated within said first process.

In another embodiment, however, after the construct is transferred the first process is suspended and the second process stored within the construct is activated, and when the second process is concluded the data and program modules of the second process are added to the first process and the first process is re-activated.

Another possibility is that after the first process terminates, a subset of data and program modules from the first process are added to the second process stored in the construct and the second process is then activated.

It may not always be necessary to transfer information relating to execution state. Therefore in another embodiment a construct is formed comprising data and/or program modules from the second process, and the construct is transferred to the first process. In this embodiment only data and program modules are transferred between processes. As in other embodiments the construct is formed by a construct operation that suspends all active threads of the second process and creates a new process comprising at least some of the data and program modules of the second process, and stores the new process in a data area of the second process. Again the construct may comprise all of the data and program modules from the second process, or may comprise only a subset of the data and program modules in which case the construct comprises only data and program modules falling within lists that are passed to the construct operation. In this embodiment the data and program modules from the second process may be copied into the first process.

When data, program modules and execution state information is being interchanged between different processes it is possible that there may be conflicts, for example between data values of two processes. Therefore preferably flagging techniques may be employed such that in the event of a conflict one process is given priority over the other such that its data, program modules and/or execution state will override those of the other process in the event of a conflict.

It should also be understood that the operations described herein are complementary and may be combined in many different ways. For example, a first process may modify itself by deletion of unwanted program modules, followed by the loading of new program modules and data, and then the resulting new process may be transferred to a second process within which it is assimilated. Equally one process may be transferred to another process and combined with it to form a new process, which then be transferred to combine with a third process and so on.

Viewed from another broad aspect it will be seen that the invention also provides a method for controlling the evolution and migration of a process comprising a set of data, program modules and execution state within a computing environment, wherein a

construct is formed comprising at least some of the data and/or program modules and execution state of a first process.

Thus it will be seen that the present invention provides techniques for the controlled evolution and migration of processes within a computing environment, which environment may be a distributed computing environment. The present invention therefore has a wide range of potential applications, including for example the creation of software applications having far greater flexibility than is known with current techniques. For example, the capability of a process is no longer fixed at compilation, but rather a process can evolve by acquiring new functionality and/or by shedding unwanted elements. Instead of transferring only data and/or program modules as with conventional object-oriented technology, a process including execution state can be transferred and new processes can be created in a semi-executed state which allows the past history of a process to be propagated. Such a technique has a number of advantages in allowing, for example, the creation of a genuine "plug-and-play" environment, and also the customisation of software in realtime without requiring recompilation or relinking.

Some embodiments of the present invention will now be described by way of example and with reference to the accompanying drawings, in which:

Fig.1 is a general schematic model of an operating environment of a computing system,

Fig.2 schematically illustrates a process life-cycle and operations that may be performed on a process,

Fig.3 is a flow-chart illustrating the operation Hibernaculum Construct(Stack s, Module m, Data d),

Fig.4 is a flow-chart illustrating operation int Assimilate(Hibernaculum h, Override Flags f),

Fig.5 is a flow-chart illustrating the operation int Usurp(Hibernaculum h, Override Flags f),

Fig.6 is a flow-chart illustrating the operation int Bequeath(Hibernaculum h, Thread threadname, Override Flags f),

Fig.7 is a flow-chart illustrating the operation int Inherit(Hibernaculum h, Thread threadname, Override Flags f),

Fig.8 is a flow-chart illustrating the operation int Mutate(Stacks, int sFlag, Module m, int mFlag, Data d, int dFlag),

Fig.9 is a flow-chart illustrating the operation int Checkpoint(Process p1, Target t), and

Fig.10 is a flow-chart illustrating the operation int Migrate(Process p1, Machine m).

Figure 1 shows the general model of a computing system. An application program 30 comprises data 10 and program modules 20. The operating system 60, also known as the virtual machine, executes the application 30 by carrying out the instructions in the program modules 20, which might cause the data 10 to be changed. The execution is effected by controlling the hardware of the underlying machine 70. The status of the execution, together with the data and results that the operating system 60 maintains for the application 30, form its execution state 40.

Such a model is general to any computing system. It should be noted here that the present invention starts from the realisation that all the information pertaining to the application at any time is completely captured by the data 10, program modules 20 and execution state 40, known collectively as the process 50 of the application 30.

The process 50 can have one or more threads of execution at the same time. Each thread executes the code of a single program module at any given time. Associated with the thread is a current context frame, which includes the following components:

- A set of registers
- A program counter, which contains the address of the next instruction to be executed
- Local variables of the module
- Input and output parameters of the module
- Temporary results of the module

In any module A, the thread could encounter an instruction to invoke another module B. In response, the program counter in the current frame is incremented, then a new context frame is created for the thread before it switches to executing module B. Upon

completing module B, the new context frame is discarded. Following that, the thread reverts to the previous frame, and resumes execution of the original module A at the instruction indicated by the program counter, i.e., the instruction immediately after the module invocation. Since module B could invoke another module, which in turn could invoke some other module and so on, the number of frames belonging to a thread may grow and reduce with module invocations and completions. However, the current frame of a thread at any given time is always the one that was created last. For this reason, the context frames of a thread are typically stored in a stack with new frames being pushed on and popped from the top. The context frames of a thread form its execution state, and the state of all the threads within the process 50 constitute its execution state 40 in Fig.1.

The data 10 and program modules 20 are shared among all threads. The data area is preferably implemented as a heap, though this is not essential. The locations of the data 10 and program modules 20 are summarized in a symbol table. Each entry in the table gives the name of a datum or a program module, its starting location in the address space, its size, and possibly other descriptors. Instead of having a single symbol table, each process may alternatively maintain two symbol tables, one for data alone and the other for program modules only, or the process could maintain no symbol table at all.

In a preferred embodiment of the present invention, the data and program code of a process are stored in a heap and a program area respectively and are shared by all the threads within the process. In addition the execution state of the process comprises a stack for each thread, each stack holding context frames, in turn each frame containing the registers, local variables and temporary results of a program module, as well as addresses for further module invocations and returns. Before describing an embodiment of the invention in more detail, however, it is first necessary to introduce some definitions of data types and functions that are used in the embodiment and which will be referred to further below.

In addition to conventional data types such as integers and pointer, four new data types Data, Module, Stack and Hibernaculum are defined in the present invention:

Data: A variable of this data type holds a set of data references. Members are added to and removed from the set by means of the following functions;

Int AddDatum(Data d, String dataname) inserts the data item dataname in the heap of the process as a member of d.

Int DelDatum(data d, String dataname) removes the data item dataname from d.

Module: A variable of this data type holds a set of references to program modules. Members are added to and removed from the set with the following functions;

Int AddModule(Module d, String modulename) inserts the program module modulename in the program area of the process as a member of d.

Int DelModule(Module d, Stringname modulename) removes the program module modulename from d.

Stack: A variable of this data type holds a list of ranges of execution frames from the stack of the threads. The list may contain frame ranges from multiple threads, however no thread can have more than one range. Variables of this type are manipulated by the following functions:

Int OpenFrame(Stack d, Thread threadname) inserts into d a new range for the thread threadname, beginning with the thread's current execution frame. This function has no effect if the thread already has a range in d.

Int CloseFrame(Stack d, Thread threadname) ends the open-ended range in d that belongs to the thread threadname. This function has no effect if the thread does not currently have an open-ended range in d.

Int PopRange(Stack d, Thread threadname) removes from d the range belonging to the thread threadname.

Hibernaculum: A variable of this data type is used to hold a suspended process.

As will be explained in more detail below a process may be suspended and stored in a construct prior to being transferred from one operating environment to another operating environment and/or may be subject to evolutionary operations:

Hibernaculum Construct(Stack s, Module m, Data d): This operation creates a new process with the execution state, program table and data heap specified as input parameters. The process is immediately suspended and then returned in a hibernaculum. The hibernaculum may be signed by the originating process as indication of its authenticity. Fig.3 is a flow-chart showing the hibernaculum construct operation.

A hibernaculum may be sent between operating environments by the following send and receive functions:

Int Send(Hibernaculum h, Target t) transmits the process contained within h to the specified target.

Hibernaculum Receive(Source s) receives from the specified source a hibernaculum containing a process.

A hibernaculum may be subject to the following evolutionary functions:

Int Assimilate(Hibernaculum h, OverrideFlags f) activates the threads of the process stored within h and runs them as threads within a calling process's operating environment. Where there is a conflict between the data and/or program modules of the hibernaculum and the operating environment, the override flags specify which to preserve. Fig.4 is a flow-chart illustrating the steps of the assimilate operation.

Int Usurp(Hibernaculum h, OverrideFlags f) copies the data and program modules of the process within h into the calling process's operating environment. Where there is a conflict between the data and/or program modules of the hibernaculum and the operating

environment, the override flags specify which to preserve. Fig.5 is a flow-chart illustrating the steps of the usurp operation.

Int Bequeath(Hibernaculum h, Thread threadname, OverrideFlags f), upon threadname's termination, activates the threads of the process stored within h and runs them as threads within the environment of the process containing threadname. Where there is a conflict between the data and/or program modules of the hibernaculum and the calling process, the override flags specify which is to be preserved. Fig.6 is a flow-chart illustrating the steps of the bequeath operation.

Int Inherit(Hibernaculum h, Thread threadname, OverrideFlags f) suspends threadname and activates the process within h. When the process within h terminates its data and program modules are added to the process containing threadname before threadname is reactivated. Where there is a conflict between the data and/or program modules of the hibernaculum and the process containing threadname, the override flags specify which is to be preserved. Fig.7 is a flow-chart illustrating the steps of the inherit operation.

Int Mutate(Stack s, int sFlag, Module m, int mflag, Data d, int dflag) modifies the execution state, program table and data heap of the calling process. If a thread has an entry in s, only the range of execution frames specified by this entry is preserved, the other frames are discarded. Execution stacks belonging to threads without an entry in s are left untouched. In addition, program modules listed in m and data items listed in d are kept or discarded depending on the flag status. Fig.8 is a flow-chart illustrating the steps of the mutate operation.

Int Checkpoint(Process p, Target t) creates a snapshot of p including all of its data, program modules and execution state. The snapshot is then sent to the specified target.

Int Migrate(Process p, Machine m) transfers the process p to the specified machine for p to continue execution there. Al of the data (including file handles and established sockets), program modules and execution state are preserved in the transfer.

The typical life cycle of a process in an embodiment of the present invention is depicted in Figure 2. First, an application 210 or a suspended process 220 stored in a hibernaculum is loaded by the operating system, then starts running as a process 230. The application 210 may be a program that is designed to perform some tasks, or it may be a simple loader that is used to start up other processes, while the suspended process 220 could be either produced locally earlier or generated on and transferred from another machine. As the process 230 goes through the application program logic, the process may create new processes, absorb other processes, mutate, or migrate as explained below.

In particular, the process may be stored in a construct hibernaculum in a suspended state. In this condition the process may be subject to a number of operations. To begin with the hibernaculum may be sent to another operating environment. Alternatively the process may be modified within its own operating environment by the selective deletion of elements from within the process, or by the selective reloading of elements into the process. A still further possibility is that the process may receive a suspended process from another operating environment, and either all or part of this suspended process may be incorporated into the first process. Alternatively all or part of the first process may be incorporated into the second process. It will also be understood that all these operations may be combined in many different ways. For example, a process may be sent from one operating environment to another and then may mutate by dropping certain elements and reloading other elements when in the second environment. In the following description the construction of a hibernaculum, send and receive functions, and exemplary evolutionary operations will all be described in greater detail.

New processes are created with the Construct 110 operation. Fig.3 is a flow-chart showing the steps of this Construct operation. Each invocation of this operation starts up a controller thread in the process 230. The controller thread freezes all other active threads in the process 230, then creates a new process with some or all of the execution state, program modules and/or data of the process 230 except for those belonging to the controller thread, before resuming the frozen threads. Therefore, the new process contain no trace of the controller thread. The new process is suspended immediately and returr in a hibernaculum in the data area of the process 230. As explained earlie

hibernaculum is a special data type that serves the sole purpose of providing a container for a suspended process. Since a process may have several hibernacula in its data area, it could create a new hibernaculum that contains those hibernacula, each of which in turn could contain more hibernacula, and so on. When the new process is activated subsequently, only those threads that were active just before the Construct 110 operation will begin to execute initially; threads that were suspended at that time will remain suspended. At the end of the Construct 110 operation, the controller thread resumes those threads that were frozen by it before terminating itself.

To specify what execution state should go into the new process, the Construct 110 operation is passed a list of ranges of context frames. The list may include frames from the state of several threads. No thread is allowed to have more than one range in this list. A thread can specify that all of its frames at the time of the Construct 110 operation are to be included in the list, by calling the AllFrame function beforehand. Alternatively, the thread can call the OpenFrame function to register its current frame, so that all the frames from the registered frame to the current frame at the time of the Construct 110 operation are included in the list. The thread can also call the CloseFrame function subsequently, to indicate that only frames from that registered with OpenFrame to the current frame at the time of calling CloseFrame are to the included in the list for the Construct 110 operation. An AllFrame or OpenFrame request for a thread erases any previous AllFrame, OpenFrame and CloseFrame requests for that thread. A CloseFrame request overrides any earlier CloseFrame request for the same thread, but the request is invalid if the thread has not already had an OpenFrame request. A thread can also make AllFrame, OpenFrame and/or CloseFrame requests on behalf of another thread by providing the identity of that thread in the requests; the effect is as if that thread is making those requests itself.

The Construct 110 operation can also be passed a list of program modules that should go into the newly created process. A thread can specify that all modules of the process 230 are to be included in the list, by calling the AllModules function prior to the Construct 110 operation. Alternatively, the thread can call the AddModule function to add a specific module to the list, and the DelModule function to remove a specific module from the list. The effect of the AllModules, AddModule and DelModule requests,

possibly made by different threads, are cumulative. Hence a DelModule request after an AllModules request would leave every module in the list except for the one removed explicitly, and a DelModule can be negated with an AddModule or AllModules request. As there could be multiple AddModule requests for the same module and AllModules could be called multiple times, a program module may be referenced several times in the list. However, the Construct 110 operation consolidates the entries in the list, so no program module gets duplicated in the new process.

To copy some or all of the data of the process 230 to the new process, the Construct 110 operation can be passed a data list. This list contains only the name of, or reference to data that should be copied. The actual data content or values that get copied to the new process are taken at the time of the Construct 110 operation, not at the time that each datum is added to the list. To ensure consistency among data that could be related to each other, all the threads in the process 230 are frozen during the Construct 110 operation. A thread can specify that all data of the process 230 are to be included in the list, by calling the AllData function prior to the Construct 110 operation. Alternatively, the thread can call the AddDatum function to add a specific datum to the list, and the DelDatum function to remove a specific datum from the list. The effect of the AllData, AddDatum and DelDatum requests, possibly made by different threads, are cumulative. Hence a DelDatum request after an AllData request would leave all of the data in the list except for the one removed explicitly, and a DelDatum can be negated with an AddDatum or AllData request. As there could be multiple AddDatum requests for the same datum and AllData could be called multiple times, a datum may be referenced several times in the list. However, the Construct 110 operation consolidates the entries in the list, so no datum gets duplicated in the new process.

Since the lists passed to the Construct 110 operation are constructed from the execution state, program modules and data of the process 230, the new process initially does not contain any component that is not found in the process 230. Consequently, the symbol table in the new process is a subset of the symbol table of the process 230. Threads in the process 230 that do not have any frame in the new process are effectively dropped from it. For those threads that have frames in the new process, when activated later, each will begin execution at the instruction indicated by the program counter in the

most recent frame amongst its frames that are copied. By excluding one or more of the most recent frames from the new process, the associated thread can be forced to return from the most recent module invocations. An exception is raised to alert the thread that those modules are not completed normally. Alternatively, the thread can be made to redo those modules upon activation, by decrementing the program counter in the most recent frame amongst those frames belonging to that thread that are copied. Similarly, by excluding one or more of its oldest frames from the new process, a thread can be forced to terminate directly after completing the frames that are included.

Hibernacula, produced by the Construct 110 operation, can be exchanged between processes via the Send 120 and Receive 130 operations. The process 230 can send a hibernaculum in its data area out on an output stream, by invoking the Send 120 operation with the hibernaculum and output stream as parameters. The output stream could lead to either a disk file, a memory stream, a device stream, or the input stream of another process, possibly on a different machine. In the former case, the process in the hibernaculum is stored away for the time being, whereas in the latter case the process in the hibernaculum is received directly into the data area of another process.

The process 230 can also acquire other processes via the Receive 130 operation, which receives from an input stream a hibernaculum containing a suspended process. The source of the input stream could be a disk file, a memory stream, a device stream, or the output stream of another process, possibly from a different machine.

The process 230 can absorb the suspended process within a hibernaculum through the Assimilate 140, Usurp 150, Bequeath 160 and Inherit 170 operations. These operations can be invoked by any thread in the process 230, or by another process that has appropriate permissions. The hibernaculum could have been created by the process 230 itself earlier, or received from another process via a disk file or an input stream.

The Assimilate 140 operation accepts a hibernaculum as input. Fig.4 is a flow-chart showing this operation in detail. The operation starts up a controller thread in the process 230. The controller thread freezes all other active threads in the process 230, adds the program modules and data of the process in the hibernaculum to the program modules and data of the process 230, respectively, and updates its symbol table accordingly. In case of a name conflict, the program module or data from the hibernaculum is discarded

in favor of the one from the process 230 by default. However, a flag can be supplied to give preference to the hibernaculum. Moreover, the context frames of the threads within the hibernaculum are added to the execution state of the process 230, enabling those threads to run within the process. Only those threads that were active just before the hibernaculum was created are activated initially; threads that were suspended at that time remain suspended. Each newly acquired thread begins execution at the instruction indicated by the program counter in the most recent frame amongst the context frames that belong to that thread. Finally, all the original threads in the process 230 that were frozen by the controller thread are resumed before it terminates itself.

The Usurp 150 operation, which also accepts a hibernaculum as input, enables the process 230 to take in only program modules and data from a hibernaculum, without acquiring its threads. Fig.5 is a flow-chart showing this operation in detail. The operation starts up a controller thread in the process 230. The controller thread freezes all other active threads in the process 230, adds the program modules and data of the process in the hibernaculum to the program modules and data of the process 230, respectively, and updates its symbol table accordingly. In case of a name conflict, the program module or data from the hibernaculum is discarded in favor of the one from the process 230 by default. However, a flag can be supplied to give preference to the hibernaculum. Finally, all the original threads in the process 230 that were frozen by the controller thread are resumed before it terminates itself.

The Bequeath 160 operation accepts a hibernaculum and a thread reference as input. Fig.6 is a flow-chart showing this operation in detail. It starts up a bequeath-thread in the process 230. The bequeath-thread registers the referenced thread and any existing bequeath-threads on that thread, then allows them to carry on execution without change. After all those threads and threads that they in turn activate have terminated, the bequeath-thread loads in the context frames, program modules and data in the hibernaculum. In case of a name conflict, the program module or data from the hibernaculum is discarded in favor of the one from the process 230 by default. However, a flag can be supplied to give preference to the hibernaculum. Subsequently, threads within the hibernaculum are activated to run in the process 230 before the bequeath-thread terminates itself. Only those threads that were active just before the hibernaculum

was created are activated initially; threads that were suspended at that time remain suspended. Each newly acquired thread begins execution at the instruction indicated by the program counter in the most recent frame amongst the context frames that belong to that thread. If multiple Bequeath 160 requests are issued for the same thread, there will be several bequeath-threads in the process 230. Each bequeath-thread will wait for all the existing bequeath-threads on the same thread, together with all the threads that they activate, to terminate before performing its function. As a result, the Bequeath 160 requests are queued up and serviced in chronological order.

The reverse of Bequeath 160 is the Inherit 170 operation, which also accepts a hibernaculum and a thread reference as input. The flow-chart for this operation is shwon in Fig.7. This operation starts up an inherit-thread in the process 230. The inherit-thread freezes the referenced thread in the process 230 together with the bequeath-threads and any inherit-thread for the referenced thread, adds the program modules and data in the hibernaculum to the program modules and data of the process 230, respectively, and updates its symbol table accordingly. In case of a name conflict, the program module or data from the hibernaculum is discarded in favor of the one from the process 230 by default. However, a flag can be supplied to give preference to the hibernaculum. Moreover, the context frames of the threads within the hibernaculum are added to the execution state of the process 230, enabling those threads to run within the process. Only those threads that were active just before the hibernaculum was created are activated initially; threads that were suspended at that time remain suspended. Each newly acquired thread begins execution at the instruction indicated by the program counter in the most recent frame amongst the context frames that belong to that thread. After all the acquired threads and threads that they in turn activate have terminated, the inherit-thread resumes those threads that were frozen by it earlier, before terminating itself. If one of the acquired threads issues an Inherit 170 request, the acquired threads and the current inherit-thread would in turn be suspended, pending the completion of the latest Inherit operation. If an acquired thread does a Bequeath 160, the inherit-thread would wait for the Bequeath request to be satisfied before resuming the threads that were frozen by it.

Besides constructing and absorbing new processes, the process 230 can also modify any of its components directly by calling the Mutate 180 operation from any thread. Fig.8

shows the flow-chart for the Mutate operation. The operation starts up a controller thread in the process 230. The controller thread first freezes all other active threads in the process 230, then selectively retains or discards its execution state, program modules and data. A list of context frames, together with a flag, can be passed to the Mutate 180 operation to retain or discard the frames in the list. Similarly, a program module list and/or a data list can be provided to indicate program modules and data of the process 230 that should be retained or discarded. The generation of the execution state, program module and data lists are the same as for the Construct 110 operation. After mutation, the controller thread resumes the threads that were frozen by it before terminating itself. Threads that no longer have a context frame in the process 230 are terminated. Each of the remaining threads resumes execution at the instruction indicated by the program counter in the most recent frame amongst the retained frames belonging to that thread. By discarding one or more of its most recent frames, a thread can be forced to return from the most recent module invocations. An exception is raised to alert the thread that those modules are not completed normally. Similarly, by discarding one or more of its oldest frames, a thread can be forced to terminate directly after completing the frames that are retained. Space freed up from the discarded context frames, program modules and data is automatically reclaimed by a garbage collector.

The process 230 is also capable of suspending and migrating itself. The Checkpoint 190 operation starts up a controller thread in the process 230. Fig.9 is a flow-chart showing the Checkpoint operation in detail. The controller thread freezes all other active threads in the process 230, then sends it in the form of a hibernaculum on a specified output stream that it established earlier. The hibernaculum contains all of the execution state, program modules, and data of the process 230, except for those belonging to the controller thread and the output stream used by the Checkpoint 190 operation. The output stream could lead to either a disk file or the input stream of another process, possibly on a different machine. The controller thread ends by terminating the process 230. When the process in the hibernaculum is activated subsequently, only those threads that were active just before the Checkpoint 190 operation will begin to execute initially; threads that were suspended at that time will remain suspended. The Checkpoint 190 operation could be

invoked by one of the threads of the process 230, or by another process with appropriate permissions.

The Migrate 200 operation is used to move the process 230 to another machine. Fig.10 is a flow-chart showing the Migrate operation in detail. The operation can be invoked by one of the threads of the process 230, or by another process with appropriate permissions. The Migrate 200 operation starts up a first controller thread in the process 230, which carries out the following steps: (a) It freezes all other active threads in the process 230. (b) It initiates a receiver process on the target machine, which runs a second controller thread. (c) The two controller threads establish an output stream in the process 230 that leads to an input stream in the receiver process. (d) The second controller thread in the receiver process performs a Receive 130 operation on its input stream. (e) The first controller thread sends the process 230 in the form of a hibernaculum on the output stream. The hibernaculum contains all of the execution state, program modules, and data of the process 230 except for those belonging to the controller thread and the output stream used for the migration. (f) The first controller thread ends by terminating the process 230. (g) After receiving the hibernaculum, the second controller thread activates the process in the hibernaculum before terminating itself. Only those threads that were active just before the migration are activated initially; threads that were suspended at that time remain suspended.

To implement the process migration and adaptation system of the present invention in a Java environment, a package called snapshot is introduced. This package contains the following classes, each of which defines a data structure that is used in the migration and adaptation operations:

```
public class Hibernaculum {
        ...
}

public class State {
        ...
}
```

```
public class Module {
        ...
}

public class Data {
        ...
}

public class Machine {
        ...
}
```

In addition, the package contains a Snapshot class that defines the migration and adaptation operations:

```
public class Snapshot {
        private static native void registerNatives();
        static {
                registerNatives();
        }

        public static native Hibernaculum Construct(State s, Module m, Data d);
        public static native int Send(Hibernaculum h, OutputStream o);
        public static native Hibernaculum Receive(InputStream i);
        public static native int Assimilate(Hibernaculum h, int f);
        public static native int Usurp(Hibernaculum h, int f);
        public static native int Bequeath(Hibernaculum h, int f);
        public static native int Inherit(Hibernaculum h, int f);
        public static native int Mutate(State s, Module m, int mflag, Data d, int dflag);
        public static native int Checkpoint(OutputStream o);
```

```
        public static native int Migrate(Machine m);

        // This class is not to be instantiated
        private Snapshot() {
        }
}
```

The methods in the Snapshot class can be invoked from application code. For example:

```
        try {
                if (snapshot.Snapshot.Construct(s, m, d) != null) {
                        // hibernaculum has been created
                } else {
                        // failed to create hibernaculum
                }
                catch(snapShot.SnapshotException e) {
                        // Failed to create hibernaculum
                }
```

The migration and adaptation operations are implemented as native codes that are added to the Java virtual machine itself, using the Java Native Interface (JNI). To do that, a Java-to-native table is first defined:

```
#define KSH "Ljava/snapshot/Hibernaculum;"
#define KSS "Ljava/snapshot/State;"
#define KSM "Ljava/snapshot/Module;"
#define KSD "Ljava/snapshot/Data;"

static JNINativeMethod snapshot_Snapshot_native_methods[] = {
        {
```

```
        "Construct",
        "("KSSKSMKSD")"KSH,
        (void*)Impl_Snapshot_Construct
    },
    {
        "Send",
        "("KSH"Ljava/io/OutputStream;)I",
        (void*)Impl_Snapshot_Send
    },
    {
        "Receive",
        "(Ljava/io/InputStream;)"KSH,
        (void*)Impl_Snapshot_Receive
    },
    {
        "Assimilate",
        "("KSH"I)I",
        (void*)Impl_Snapshot_Assimilate
    },
    {
        "Usurp",
        "("KSH"I)I",
        (void*)Impl_Snapshot_Usurp
    },
    {
        "Bequeath",
        "("KSH"I)I",
        (void*)Impl_Snapshot_Bequeath
    },
    {
        "Inherit",
```

```
            "("KSH"I)I",
            (void*)Impl_Snapshot_Inherit
        },
        {
            "Mutate",
            "("KSSKSM"I"KSD"I)I",
            (void*)Impl_Snapshot_Mutate
        },
        {
            "Checkpoint",
            "(Ljava/io/OutputStream;)I",
            (void*)Impl_Snapshot_Checkpoint
        },
        {
            "Migrate",
            "(Ljava/snapshot/Machine;)I",
            (void*)Impl_Snapshot_Migrate
        },
};
```

After that, the native implementations are registered via the following function:

```
JNIEXPORT void JNICALL
Java_snapshot_Snapshot_registerNatives(JNIEnv *env, jclass cls) {
        (*env)->RegisterNatives(    env,
                                    cls,
                                    snapshot_Snapshot_native_methods,
                                    sizeof(snapshot_Snapshot_native_methods) /
                                    sizeof(JNINativeMethod)     );
}
```

Besides the above native codes, several functions are added to the Java virtual machine implementation, each of which realizes one of the migration and adaptation operations:

```
void* Impl_Snapshot_Construct(..) {
        // follow flowchart in Figure 3
        ...
}

void* Impl_Snapshot_Send(..) {
        // send given hibernaculum to specified target
        ...
}

void* Impl_Snapshot_Receive(..) {
        // receive a hibernaculum from a specified source
        ...
}

void* Impl_Snapshot_Assimilate(..) {
        // follow flowchart in Figure 4
        ...
}

void* Impl_Snapshot_Usurp(..) {
        // follow flowchart in Figure 5
        ...
}

void* Impl_Snapshot_Bequeath(..) {
        // follow flowchart in Figure 6
```

```
        ...
}

void* Impl_Snapshot_Inherit(..) {
        // follow flowchart in Figure 7
        ...
}

void* Impl_Snapshot_Mutate(..) {
        // follow flowchart in Figure 8
        ...
}

void* Impl_Snapshot_Checkpoint(..) {
        // follow flowchart in Figure 9
        ...
}

void* Impl_Snapshot_Migrate(..) {
        // follow flowchart in Figure 10
        ...
}
```

CLAIMS

1. A computing environment wherein a process comprising a set of data and/or program modules and execution state is treated as an entity that can be transferred between components of said environment, and wherein a said process can be subject to an evolutionary operation.

2. A computing environment as claimed in claim 1 wherein a construct is formed comprising data and/or program modules and execution state of a first process, and wherein said evolutionary operations are performed by functions operating on a said construct.

3. A computing environment as claimed in claim 2 wherein said construct is formed by a construct operation that suspends all active threads of said first process and creates a new process comprising at least some of the data and/or program modules and execution state of said first process, and stores said new process in a data area of said first process.

4. A computing environment as claimed in claim 3 wherein said construct comprises only data, program modules and execution state falling within lists that are passed to said construct operation.

5. A computing environment as claimed in any of claims 2 to 4 wherein said construct is provided with an authorising signature.

6. A computing environment as claimed in any preceding claim wherein said evolutionary operations include the selective deletion of objects from within said process.

7. A computing environment as claimed in any preceding claim wherein said evolutionary operations include the selective loading or reloading of objects into said process.

8. A computing environment as claimed in claim 1 wherein a said evolutionary operation includes the incorporation into a first process of new objects from a second process.

9. A computing environment as claimed in claim 8 wherein a construct is formed comprising at least some of the data and/or program modules and execution state from said second process, and said construct is transferred to said first process.

10. A computing environment as claimed in claim 9 wherein said construct is formed by a construct operation that suspends all active threads of said second process and creates a new process comprising a subset of the data, program modules and execution state of said second process, and stores said new process in a data area of said second process.

11. A computing environment as claimed in claim 10 wherein said construct comprises only data, program modules and execution state falling within lists that are passed to said construct operation.

12. A computing environment as claimed in any of claims 9 to 11 wherein said construct is provided with an authorising signature.

13. A computing environment as claimed in any of claims 9 to 12 wherein after said construct is transferred the second process stored within said construct is caused to be activated within said first process.

14. A computing environment as claimed in any of claims 9 to 12 wherein after said construct is transferred the first process is suspended and the second process stored within said construct is activated, and when the second process is concluded the data and program modules of the second process are added to the first process and the first process is re-activated.

15. A computing environment as claimed in any of claims 9 to 12 wherein after said first process terminates at least some of the data and/or program modules from said first process are added to the second process stored in said construct and said second process is then activated.

16. A computing environment as claimed in claim 8 wherein a construct is formed comprising at least some of the data and/or program modules from said second process, and said construct is transferred to said first process.

17. A computing environment as claimed in claim 16 wherein said construct is formed by a construct operation that suspends all active threads of said second process and creates a new process comprising at least some of the data and/or program modules of said second process, and stores said new process in a data area of said second process.

18. A computing environment as claimed in claim 17 wherein said construct comprises only a subset of data and program modules falling within lists that are passed to said construct operation.

19. A computing environment as claimed in any of claims 16 to 18 wherein said data and said program modules from said second process are copied into said first process.

20. A computing environment as claimed in any of claims 8 to 19 wherein in the event of a conflict between data and/or program modules of said first process and data and/or program modules of said second process, the data and/or program modules of said first process will override the data and/or program modules of said second process.

21. A computing environment as claimed in any of claims 8 to 19 wherein in the event of a conflict between data and/or program modules of said first process and data and/or program modules of said second process, the data and/or program modules of said second process will override the data and/or program modules of said first process.

22. A computing environment as claimed in claim 1 wherein a said process may be transferred between different first and second hardware components of said computing environment.

23. A computing environment as claimed in claim 22 wherein a construct is formed comprising at least some of the data and/or program modules and execution state of said process, and said construct is transferred.

24. A computing environment as claimed in claim 22 wherein said construct comprises a subset of the data, program modules and execution state of said process.

25. A computing environment as claimed in any of claims 22 to 24 wherein a said process is subject to an evolutionary operation that allows the process to run in the second hardware component.

26. A computing environment as claimed in any of claims 22 to 24 wherein said second hardware component is a memory storage device.

27. A method for controlling the evolution and migration of a process comprising a set of data and program modules and execution state within a computing environment, wherein a construct is formed comprising data and/or program modules and execution state of a first process.

28. A method as claimed in claim 27 wherein said construct is formed by a construct operation that suspends all active threads of said first process and creates a new process comprising at least some of the data and/or program modules and execution state of said first process, and stores said new process in a data area of said first process.

29. A method as claimed in claim 28 wherein the construct comprises all the data, program modules and execution states of said first process.

30. A method as claimed in claim 28 wherein said construct comprises only data, program modules and execution state falling within lists that are passed to said construct operation.

31. A method as claimed in any of claims 28 to 30 wherein said construct is provided with an authorising signature.

32. A method as claimed in any of claims 27 to 31 wherein a process is modified by the selective deletion of objects from within the process.

33. A method as claimed in any of claims 27 to 31 wherein a process is modified by the selective loading or reloading of objects into a said process.

34. A method as claimed in claim 27 comprising incorporating into a first process objects from a second process.

35. A method as claimed in claim 34 wherein a construct is formed comprising at least some of the data and/or program modules and execution state from said second process, and said construct is transferred to said first process.

36. A method as claimed in claim 35 wherein said construct is formed by a construct operation that suspends all active threads of said second process and creates a new process comprising some of the data and/or program modules and execution state of said second process, and stores said new process in a data area of said second process.

37. A method as claimed in claim 36 wherein said construct comprises only data, program modules and execution states falling within lists that are passed to said construct operation.

38. A method as claimed in any of claims 35 to 37 wherein said construct is formed with an authorising signature.

39. A method as claimed in any of claims 35 to 38 wherein after said construct is transferred the second process stored within said construct is caused to be activated within said first process.

40. A method as claimed in any of claims 35 to 38 wherein after said construct is transferred the first process is suspended and the second process stored within said construct is activated, and when the second process is concluded the data and program modules of the second process are added to the first process and the first process is re-activated.

41. A method as claimed in any of claims 35 to 38 wherein after said first process terminates at least some of the data and/or program modules from said first process are added to the second process stored in said construct and said second process is then activated.

42. A method as claimed in claim 34 wherein a construct is formed comprising at least some of the data and/or program modules from said second process, and said construct is transferred to said first process.

43. A method as claimed in claim 42 wherein said construct is formed by a construct operation that suspends all active threads of said second process and creates a new process comprising some of the data and program modules of said second process, and stores said new process in a data area of said second process.

44. A method as claimed in claim 43 wherein said construct comprises only data and program modules falling within lists that are passed to said construct operation.

45. A method as claimed in any of claims 42 to 44 wherein said at least some data and/or program modules from said second process are copied into said first process.

46. A method as claimed in any of claims 34 to 45 wherein in the event of a conflict between data and/or program modules of said first process and data and/or program modules of said second process, the data and/or program modules of said first process will override the data and/or program modules of said second process.

47. A method as claimed in any of claims 34 to 45 wherein in the event of a conflict between data and/or program modules of said first process and data and/or program modules of said second process, the data and/or program modules of said second process will override the data and/or program modules of said first process.

48. A method as claimed in claim 27 wherein a said process is transferred between different first and second hardware components of said computing environment.

49. A method as claimed in claim 48 wherein a construct is formed comprising at least some of the data and/or program modules and execution state of said process, and said construct is transferred.

50. A method as claimed in claim 49 wherein said construct comprises a subset of the data, program modules and execution state of said process.

51. A method as claimed in any of claims 48 to 50 wherein a said process is subject to an evolutionary operation that allows the process to run in the second hardware component.

52. A method as claimed in any of claims 48 to 50 wherein said second hardware component is a memory storage device.

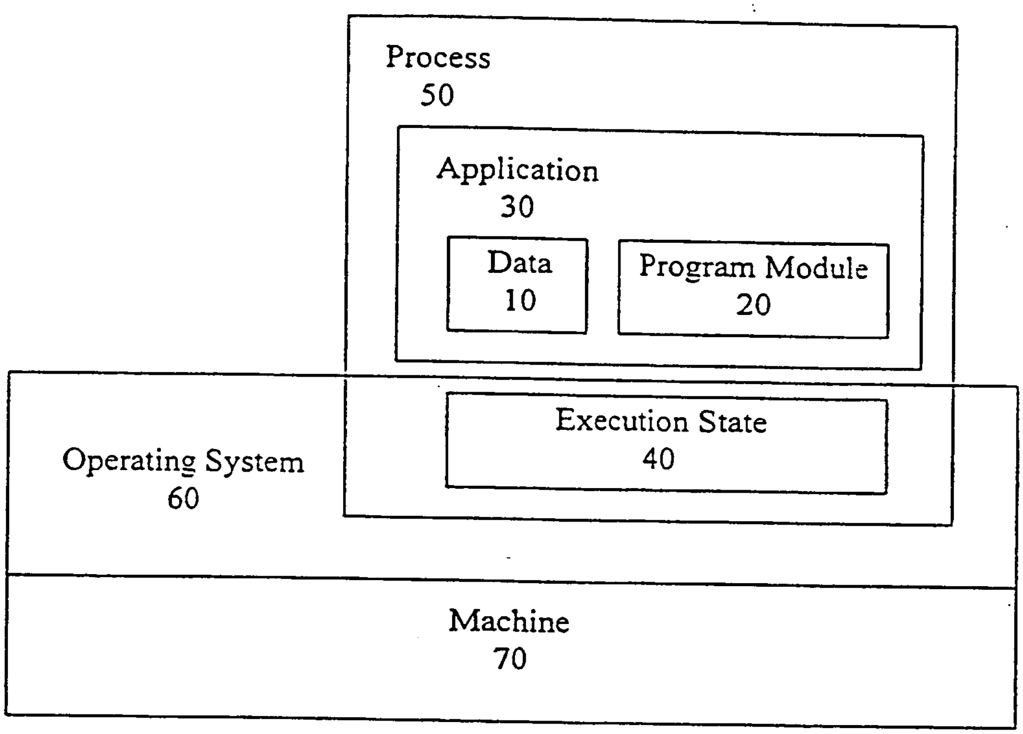

Process
50
Application
30
Data
10
Program Module
20
Execution State
40
Operating System
60
Machine
70

Fig. 1

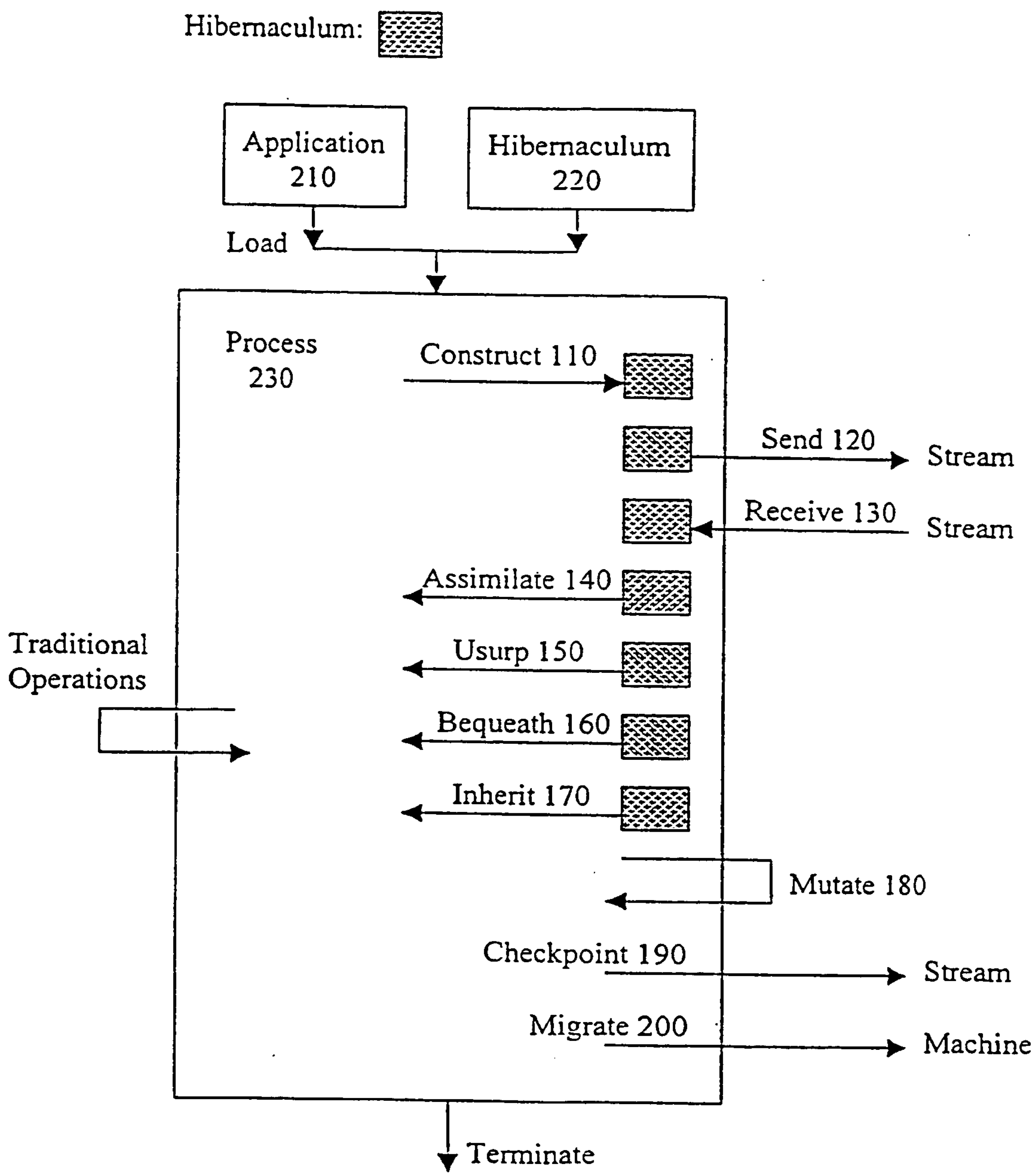

Hibernaculum:
Application 210
Hibernaculum 220
Load
Process 230
Construct 110
Send 120
Stream
Receive 130
Stream
Assimilate 140
Traditional Operations
Usurp 150
Bequeath 160
Inherit 170
Mutate 180
Checkpoint 190
Stream
Migrate 200
Machine
Terminate

Fig. 2

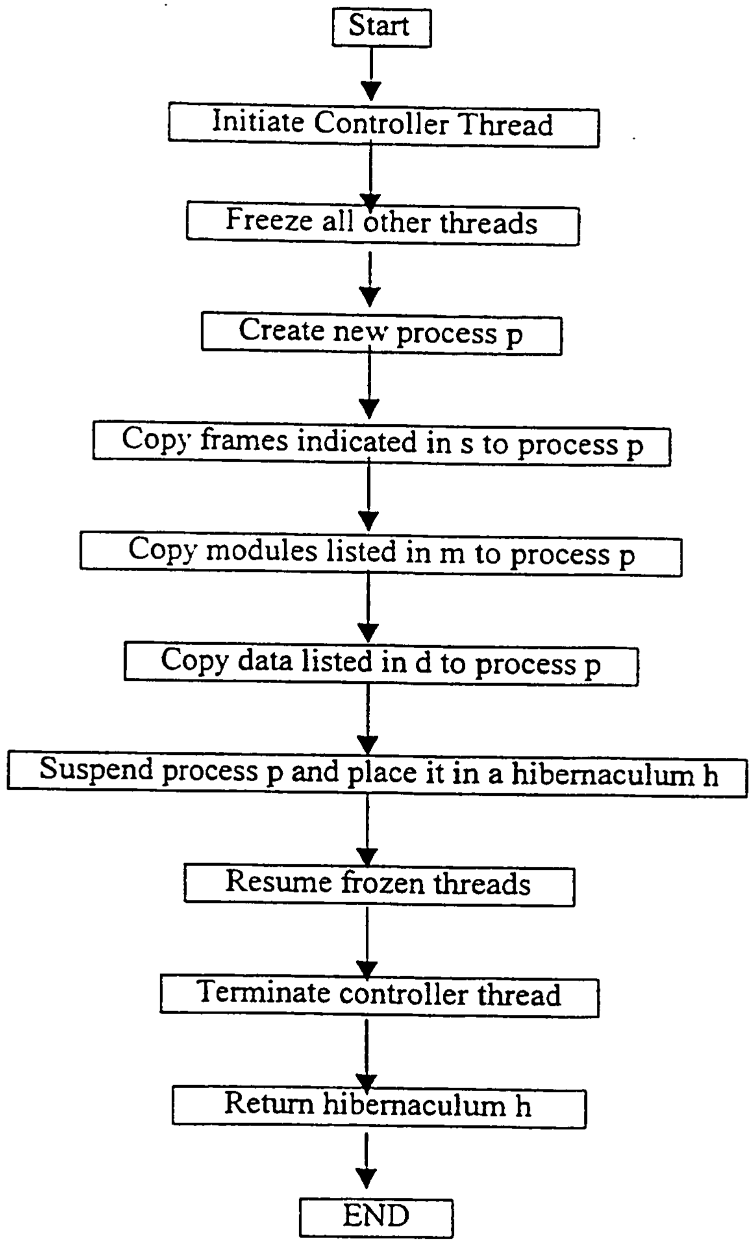

Start
Initiate Controller Thread
Freeze all other threads
Create new process p
Copy frames indicated in s to process p
Copy modules listed in m to process p
Copy data listed in d to process p
Suspend process p and place it in a hibernaculum h
Resume frozen threads
Terminate controller thread
Return hibernaculum h
END

Fig. 3

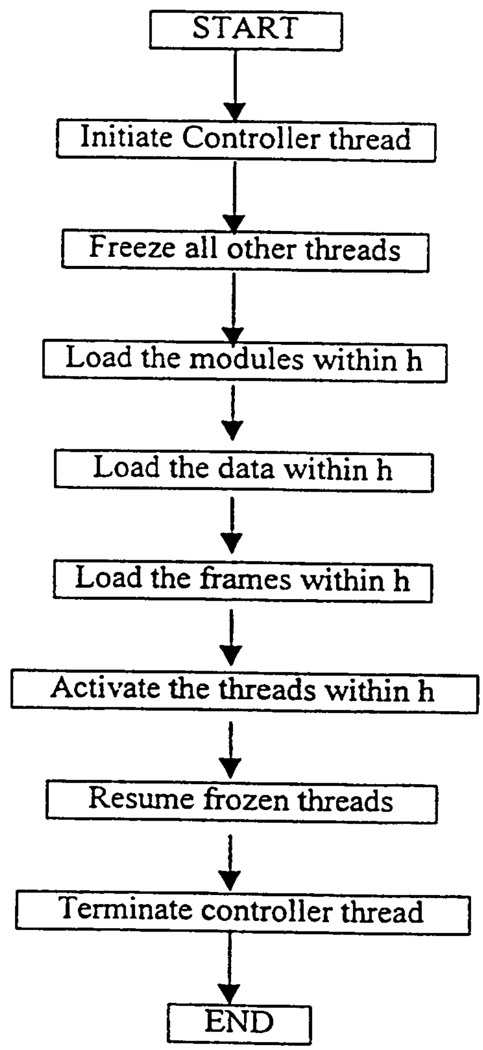

START
Initiate Controller thread
Freeze all other threads
Load the modules within h
Load the data within h
Load the frames within h
Activate the threads within h
Resume frozen threads
Terminate controller thread
END

Fig. 4

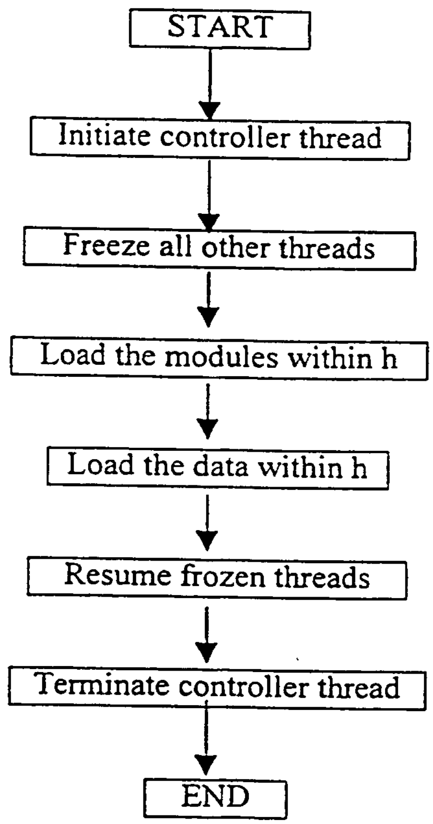

START
Initiate controller thread
Freeze all other threads
Load the modules within h
Load the data within h
Resume frozen threads
Terminate controller thread
END

Fig. 5

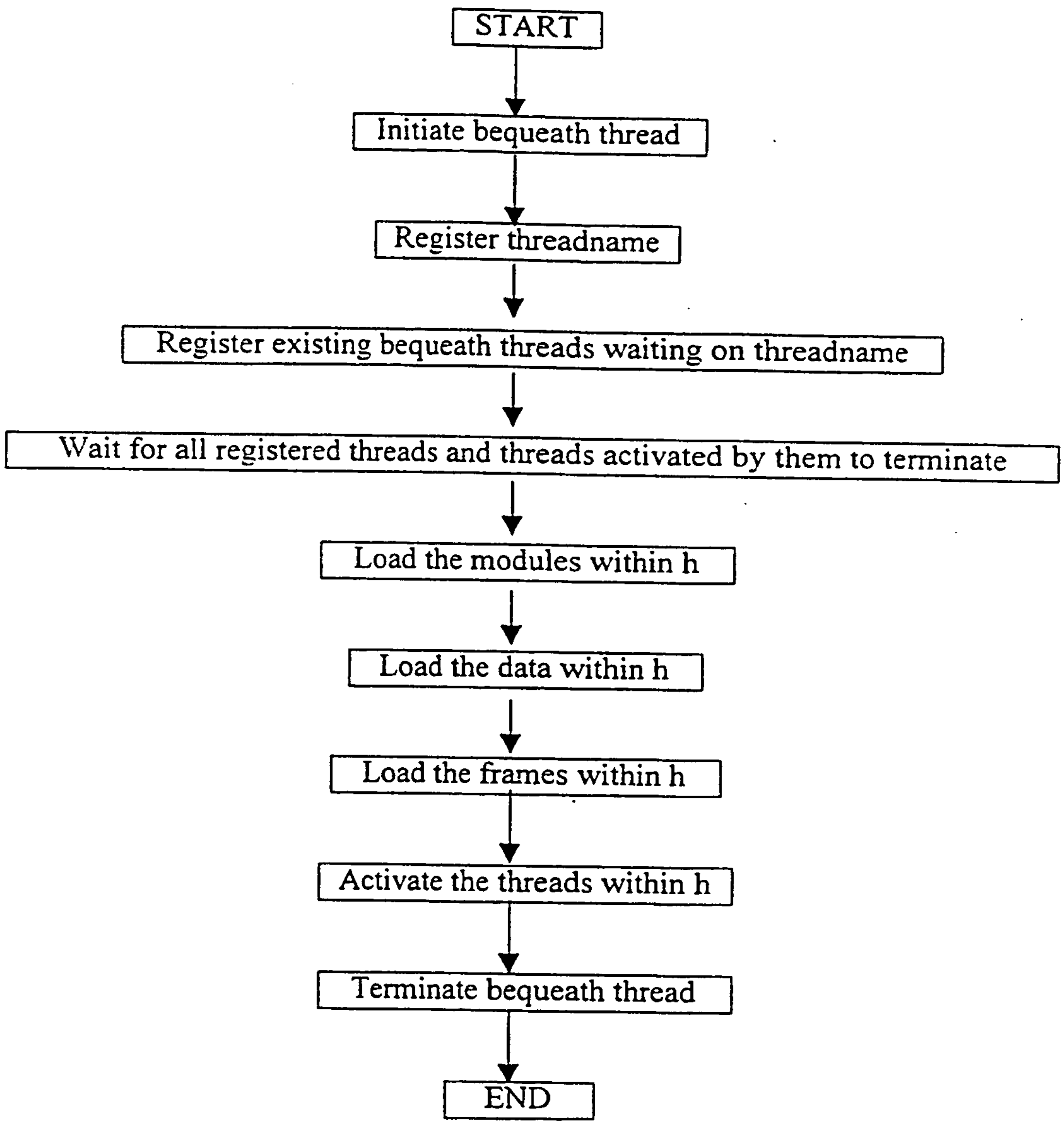

START
Initiate bequeath thread
Register threadname
Register existing bequeath threads waiting on threadname
Wait for all registered threads and threads activated by them to terminate
Load the modules within h
Load the data within h
Load the frames within h
Activate the threads within h
Terminate bequeath thread
END

Fig. 6

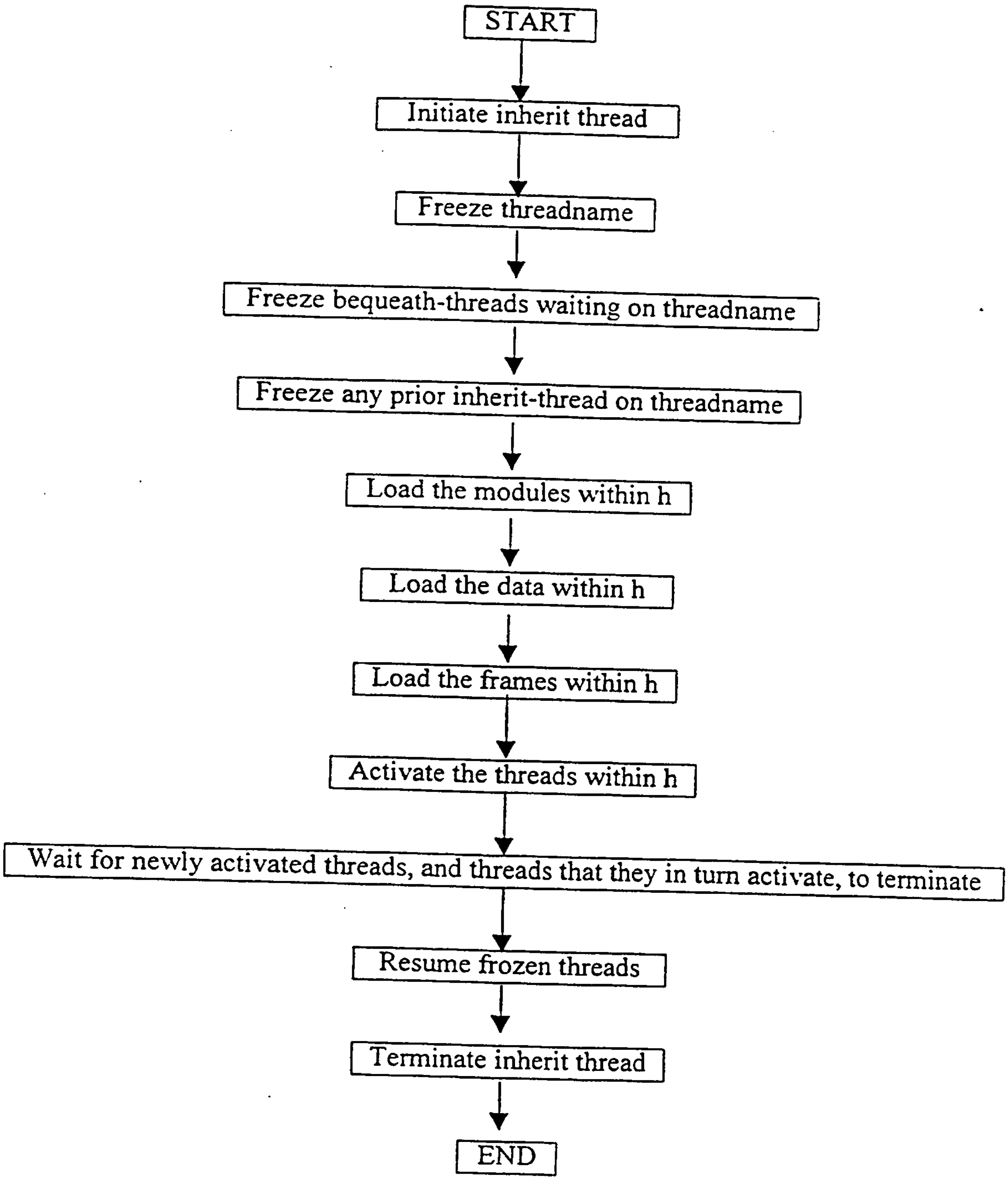

START
Initiate inherit thread
Freeze threadname
Freeze bequeath-threads waiting on threadname
Freeze any prior inherit-thread on threadname
Load the modules within h
Load the data within h
Load the frames within h
Activate the threads within h
Wait for newly activated threads, and threads that they in turn activate, to terminate
Resume frozen threads
Terminate inherit thread
END

Fig. 7

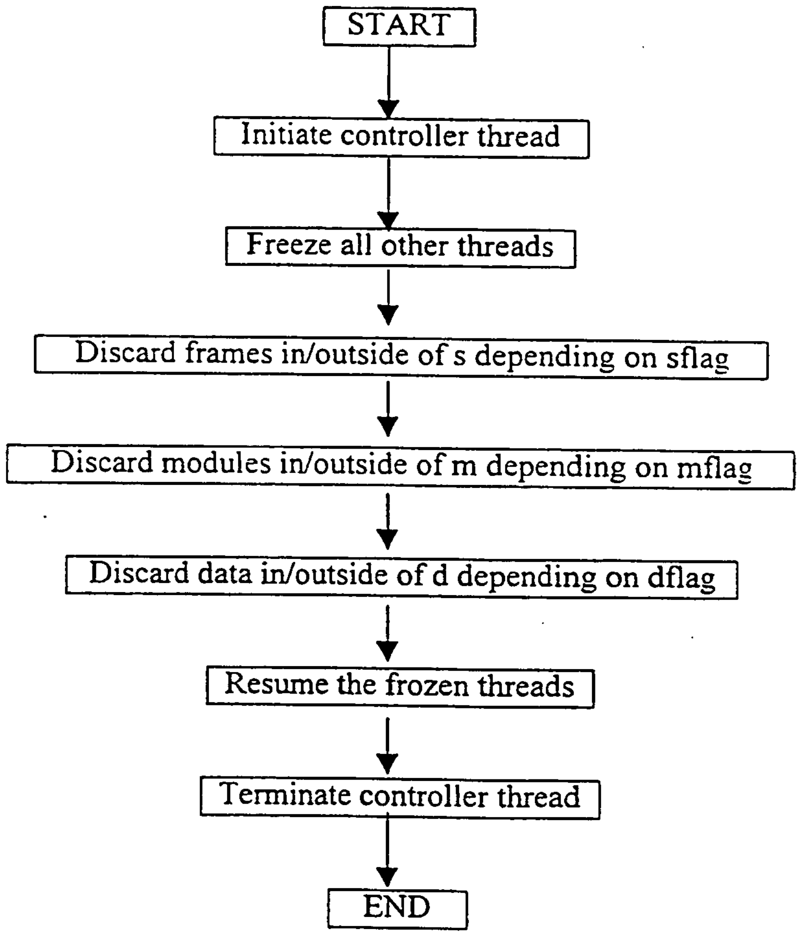

START
Initiate controller thread
Freeze all other threads
Discard frames in/outside of s depending on sflag
Discard modules in/outside of m depending on mflag
Discard data in/outside of d depending on dflag
Resume the frozen threads
Terminate controller thread
END

Fig. 8

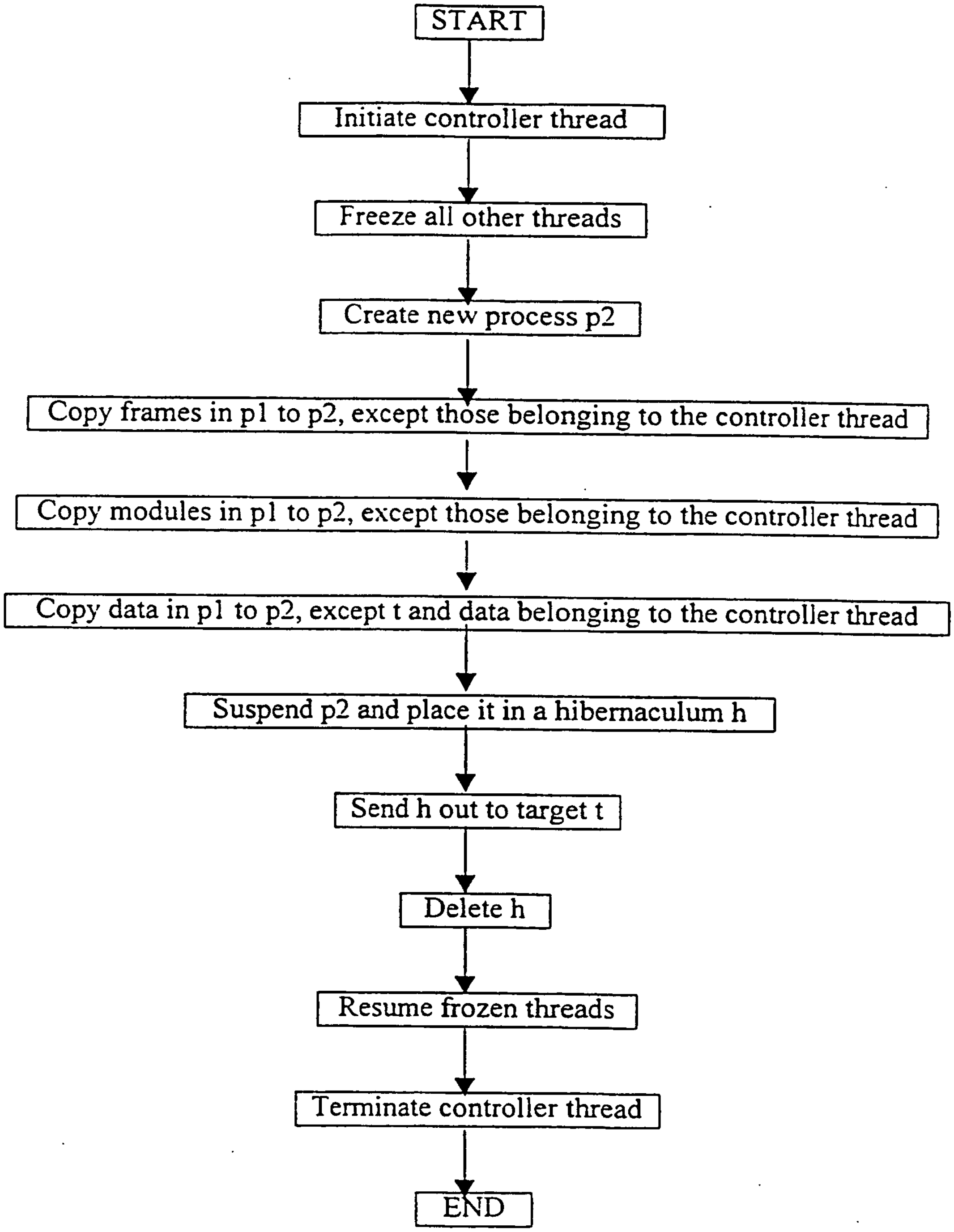

START
Initiate controller thread
Freeze all other threads
Create new process p2
Copy frames in p1 to p2, except those belonging to the controller thread
Copy modules in p1 to p2, except those belonging to the controller thread
Copy data in p1 to p2, except t and data belonging to the controller thread
Suspend p2 and place it in a hibernaculum h
Send h out to target t
Delete h
Resume frozen threads
Terminate controller thread
END

Fig. 9

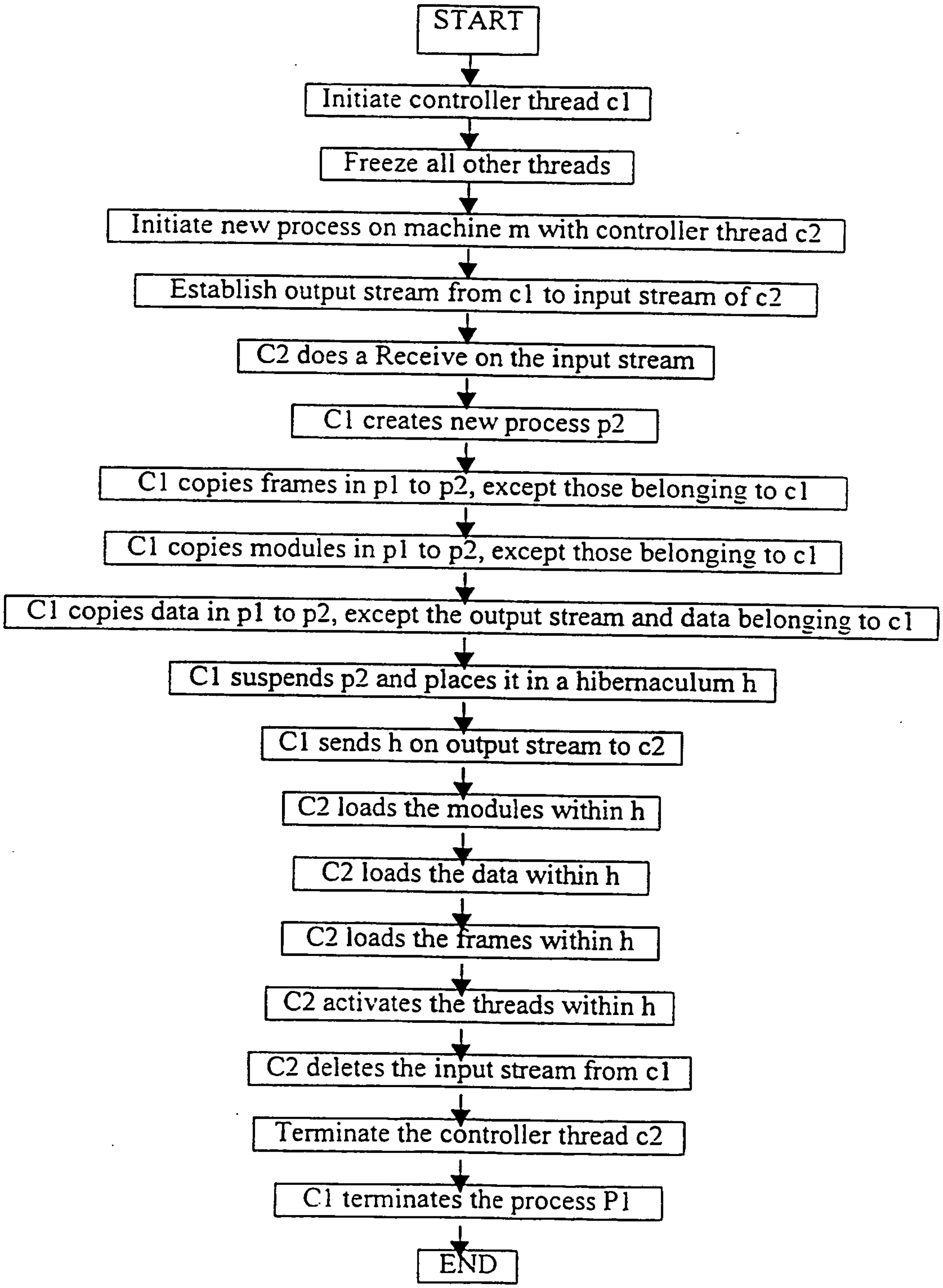

START
Initiate controller thread c1
Freeze all other threads
Initiate new process on machine m with controller thread c2
Establish output stream from c1 to input stream of c2
C2 does a Receive on the input stream
C1 creates new process p2
C1 copies frames in p1 to p2, except those belonging to c1
C1 copies modules in p1 to p2, except those belonging to c1
C1 copies data in p1 to p2, except the output stream and data belonging to c1
C1 suspends p2 and places it in a hibernaculum h
C1 sends h on output stream to c2
C2 loads the modules within h
C2 loads the data within h
C2 loads the frames within h
C2 activates the threads within h
C2 deletes the input stream from c1
Terminate the controller thread c2
C1 terminates the process P1
END

Fig. 10

INTERNATIONAL SEARCH REPORT

International application No.
PCT / SG 98/00102

A. CLASSIFICATION OF SUBJECT MATTER

IPC^{6}: G06F 9/46, G06F 9/54

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC^{6}: G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPODOC, WPI, The INTERNET, ACM Digital Library

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y	US 5603031 A (HELGESON et al.), 11 February 1997 (11.02.77) summary; column 7, line 66 - column 8, line 10; column 8, line 51 ff.; column 10, line 31- column 11, line 28; fig. 24A-C; totality.	1-5,22-25,27-31,48-51 5,12,31,38
X	Aptiva Benutzerhandbuch. MK NLS Center, Kst. 2877, June 1996, which is a translation of the Aptiva Handbook, IBM Nr. 07H1639, International Business Machines Corporation "Hinweise zu den Rapid Resume Funktionen".	1-3,22,23,26-29,48,49,52
X	Rapid Resume. [online], [retrieved on 2000-02-16]. Retrieved from the Internet URL: <http://servicepac. mainz.ibm.com /eprmhtml /eprm3/7732.htm>; totality.	1-3,22,23,26-29,48,49,52

☒ Further documents are listed in the continuation of Box C. ☒ See patent family annex.

* Special categories of cited documents:
„A" document defining the general state of the art which is not considered to be of particular relevance
„E" earlier application or patent but published on or after the international filing date
„L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
„O" document referring to an oral disclosure, use, exhibition or other means
„P" document published prior to the international filing date but later than the priority date claimed

„T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
„X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
„Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
„&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
19 May 2000 (19.05.00)	22 May 2000 (22.05.00)
Name and mailing adress of the ISA/AT Austrian Patent Office Kohlmarkt 8-10; A-1014 Vienna Facsimile No. 1/53424/535	Authorized officer Fastenbauer Telephone No. 1/53424/447

INTERNATIONAL SEARCH REPORT

International application No.
PCT /SG 98 / 00102

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y	OSHIMA Mitsuru and KARJOTH Guenther. Aglets Specification (1.0), Draft 0.30 [online]1997-05-20 [retrieved on 2000-05-18]. Retrieved from the Internet URL:< http://www.trl.ibm.co.jp/aglets/ spec10.html> totality.	1-4,6-12,14,16, 19-30,32-38, 40,42,45,48-52 5,12,31,38
A	CHAUHAN Deepika and BAKER Albert D. JAFMAS: a multiagent application development system. Proceedings of the second international conference of Autonomous agents, Minneapolis, MN USA, p. 100-107 [online], May 10-13,1998 [retrieved on 2000-05-17]. Retrieved from the Internet URL:< http://www.acm.org/pubs/ citations/ proceedings /ai/280765/p100-chauhan/p100-chauhan.pdf>; totality.	1-52
A	VENNERS Bill. Solve real problems with aglets, a type of mobile agent. JavaWorld, May 1997 [online] [retrieved on 2000-05-18]. Retrieved from the Internet URL:< http: http://www.javaworld.com/ javaworld/jw-05-1997/f_jw-05-hood.html>; totality, especially Aglets, A refresher.	1-52
A	ARIDOR, Yariv, and LANGE Danny B. Agent design patterns: elements of agent application design. Proceedings of the second international conference of Autonomous agents, Minneapolis, MN USA, p.108-115 [online], May 10-13,1998 [retrieved on 2000-05-17]. Retrieved from the Internet URL:< http://www.acm.org/pubs/ articles/proceedings/ai/280765/p108-aridor/p108-aridor.pdf>; totality.	1-52
A	openUTM V4.0 (BS2000/OSD), Concepts and Functions, ID: U20683-J-Z135-2-7600 [online] February 1997 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL: http://manuals.mchp. siemens.de/servers /bs2_man /man_us/utm/v4_0/utm_kon.pdf> Chapter 5: Structure of UTM Applications, especially 5.2: The process concept, para. 3, page 67 .	1-52
A	BS2000/OSD-BC V1.0, Performance Handbook, ID: U1794-J-Z125-6-7600 [online] February 1997 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL: http://manuals.mchp.siemens.de/ servers/bs2_man/man_us/bs2_bc/V1_0/perform.pdf> Chapter 5.3.1: Managing the resource main memory, especially: Deactivation (p.248), Waiting Time runout Control (p.255),Paging management algorithms (pp.256-257).	1-52

INTERNATIONAL SEARCH REPORT

International application No.
PCT /SG 98 / 00102

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	BS2000/OSD-BC V1.0, Dynamic Binder Loader/Starter, ID: U5137-J-Z125-2-7600 [online] April 1993 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL: http://manuals.mchp.siemens.de/servers/bs2_man/man_us/bs2_bc/V1_0/bls.pdf> Chapter 2.3.1: Unloading and unlinking objects (p.44).	1-52
A	WO 97/35262 A (HITACHI), 25 September 1997 (25.09.97), summary.	1-52

- - - -

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SG 98/00102

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
   because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
   because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
   because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This international Searching Authority found multiple inventions in this international application, as follows:

see extra sheet

1. ☒ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying and additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

new				
1	cl. 1-4	+	cl. 27-28 cl. 30	a computing environment (method) wherein a process ... is treated as an entity that can be transferred ... a construct is formed ... / suspend all active threads ... falling whithin lists
2			cl. 29	... comprises all the data ...
3	cl. 5	+	cl. 31	... authorizing signature
4	cl. 6	+	cl. 32	... selective deletion of objects ...
5	cl. 7	+	cl. 33	... selective loading / Reloading of objects ...
6	cl. 8	+	cl. 34	... includes ... the incorporation ... of new objects ...
	cl. 9	+	cl. 35	... construct ... comprising at least ... data and/or program modules ... transferred to said first process ... stores said new process ...
	cl. 10	+	cl. 36	... suspends all active threads ... and creates an new process ...
	cl. 11	+	cl. 37	... falling within lists ...
	cl. 12	+	cl. 38	... authorizing signature
	cl. 13	+	cl. 39	... after said construct is transferred ... the second process is caused to be activeted ...
	cl. 14	+	cl. 40	... the first process is suspended and the second ... activated ... the first re-activated ...
	cl. 15	+	cl. 41	... some of the data / modules ... are added to the second process ..
	cl. 16	+	cl. 42	... a construct is formed ... and transferred to said first process
	cl. 17	+	cl. 43	... suspends all active threads of said second process and creates a new process ...
	cl. 18	+	cl. 44	... whithin lists...
	cl. 19	+	cl. 45	... said data and said program modules are copied into said first process
7	cl. 22	+	cl. 48	said process may be transferred ...
	cl. 23	+	cl. 49	... a construct is formed at least some of the data and/or program modules and executurion states
	cl. 24	+	cl. 50	... a subset of the data, program modules and execution states ...
	cl. 25	+	cl. 51	... allows to run in the second hardware ...
	cl. 26	+	cl. 52	... is a memory storage device ...

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/SG 98/00102

Patent document cited in search report			Publication date	Patent family member(s)			Publication date
US	A	5603031	11-02-1997	AU	A1	72164/94	06-02-1995
				EP	A2	634719	18-01-1995
				EP	A3	634719	03-01-1996
				JP	A2	7182174	21-07-1995
				JP	T2	7509799	26-10-1995
				WO	A1	9502219	19-01-1995
				US	A	6016393	18-01-2000
WO	A1	9735262	25-09-1997			none	

INTERNATIONAL SEARCH REPORT

International application No.
PCT / SG 98/00102

A. CLASSIFICATION OF SUBJECT MATTER

IPC^6: G06F 9/46, G06F 9/54

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC^6: G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPODOC, WPI, The INTERNET, ACM Digital Library

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y	US 5603031 A (HELGESON et al.), 11 February 1997 (11.02.77) summary; column 7, line 66 - column 8, line 10; column 8, line 51 ff.; column 10, line 31- column 11, line 28; fig. 24A-C; totality.	1-5,22-25,27-31,48-51 5,12,31,38
X	Aptiva Benutzerhandbuch. MK NLS Center, Kst. 2877, June 1996, which is a translation of the Aptiva Handbook, IBM Nr. 07H1639, International Business Machines Corporation "Hinweise zu den Rapid Resume Funktionen".	1-3,22,23,26-29,48,49,52
X	Rapid Resume. [online], [retrieved on 2000-02-16]. Retrieved from the Internet URL: <http://servicepac. mainz.ibm.com /eprmhtml /eprm3/7732.htm>; totality.	1-3,22,23,26-29,48,49,52

☒ Further documents are listed in the continuation of Box C. ☒ See patent family annex.

* Special categories of cited documents:
„A" document defining the general state of the art which is not considered to be of particular relevance
„E" earlier application or patent but published on or after the international filing date
„L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
„O" document referring to an oral disclosure, use, exhibition or other means
„P" document published prior to the international filing date but later than the priority date claimed
„T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
„X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
„Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
„&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
19 May 2000 (19.05.00)	22 May 2000 (22.05.00)
Name and mailing adress of the ISA/AT Austrian Patent Office Kohlmarkt 8-10; A-1014 Vienna Facsimile No. 1/53424/535	Authorized officer Fastenbauer Telephone No. 1/53424/447

INTERNATIONAL SEARCH REPORT

International application No.
PCT /SG 98 / 00102

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y	OSHIMA Mitsuru and KARJOTH Guenther. Aglets Specification (1.0), Draft 0.30 [online]1997-05-20 [retrieved on 2000-05-18]. Retrieved from the Internet URL:< http://www.trl.ibm.co.jp/aglets/ spec10.html> totality.	1-4,6-12,14,16, 19-30,32-38, 40,42,45,48-52 5,12,31,38
A	CHAUHAN Deepika and BAKER Albert D. JAFMAS: a multiagent application development system. Proceedings of the second international conference of Autonomous agents, Minneapolis, MN USA, p. 100-107 [online], May 10-13,1998 [retrieved on 2000-05-17]. Retrieved from the Internet URL:< http://www.acm.org/pubs/ citations/ proceedings /ai/280765/p100-chauhan/p100-chauhan.pdf>; totality.	1-52
A	VENNERS Bill. Solve real problems with aglets, a type of mobile agent. JavaWorld, May 1997 [online] [retrieved on 2000-05-18]. Retrieved from the Internet URL:< http: http://www.javaworld.com/ javaworld/jw-05-1997/f_jw-05-hood.html>; totality, especially Aglets, A refresher.	1-52
A	ARIDOR, Yariv, and LANGE Danny B. Agent design patterns: elements of agent application design. Proceedings of the second international conference of Autonomous agents, Minneapolis, MN USA, p.108-115 [online], May 10-13,1998 [retrieved on 2000-05-17]. Retrieved from the Internet URL:< http://www.acm.org/pubs/ articles/proceedings/ai/280765/p108-aridor/p108-aridor.pdf>; totality.	1-52
A	openUTM V4.0 (BS2000/OSD), Concepts and Functions, ID: U20683-J-Z135-2-7600 [online] February 1997 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL: http://manuals.mchp. siemens.de/servers /bs2_man /man_us/utm/v4_0/utm_kon.pdf> Chapter 5: Structure of UTM Applications, especially 5.2: The process concept, para. 3, page 67 .	1-52
A	BS2000/OSD-BC V1.0, Performance Handbook, ID: U1794-J-Z125-6-7600 [online] February 1997 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL: http://manuals.mchp.siemens.de/ servers/bs2_man/man_us/bs2_bc/V1_0/perform.pdf> Chapter 5.3.1: Managing the resource main memory, especially: Deactivation (p.248), Waiting Time runout Control (p.255),Paging management algorithms (pp.256-257).	1-52

INTERNATIONAL SEARCH REPORT

International application No.
PCT /SG 98 / 00102

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	
A	BS2000/OSD-BC V1.0, Dynamic Binder Loader/Starter, ID: U5137-J-Z125-2-7600 [online] April 1993 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL: http://manuals.mchp.siemens.de/servers/bs2_man/man_us/bs2_bc/V1_0/bls.pdf> Chapter 2.3.1: Unloading and unlinking objects (p.44).	1-52
A	WO 97/35262 A (HITACHI), 25 September 1997 (25.09.97), summary.	1-52

- - - -

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SG 98/00102

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
   because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
   because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
   because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This international Searching Authority found multiple inventions in this international application, as follows:

see extra sheet

1. ☒ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying and additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

new				
1	cl. 1-4	+	cl. 27-28 cl. 30	a computing environment (method) wherein a process ... is treated as an entity that can be transferred ... a construct is formed ... / suspend all active threads ... falling whithin lists
2			cl. 29	... comprises all the data ...
3	cl. 5	+	cl. 31	... authorizing signature
4	cl. 6	+	cl. 32	... selective deletion of objects ...
5	cl. 7	+	cl. 33	... selective loading / Reloading of objects ...
6	cl. 8	+	cl. 34	... includes ... the incorporation ... of new objects ...
	cl. 9	+	cl. 35	... construct ... comprising at least ... data and/or program modules ... transferred to said first process ... stores said new process ...
	cl. 10	+	cl. 36	... suspends all active threads ... and creates an new process ...
	cl. 11	+	cl. 37	... falling within lists ...
	cl. 12	+	cl. 38	... authorizing signature
	cl. 13	+	cl. 39	... after said construct is transferred ... the second process is caused to be activeted ...
	cl. 14	+	cl. 40	... the first process is suspended and the second ... activated ... the first re-activated ...
	cl. 15	+	cl. 41	... some of the data / modules ... are added to the second process ..
	cl. 16	+	cl. 42	... a construct is formed ... and transferred to said first process
	cl. 17	+	cl. 43	... suspends all active threads of said second process and creates a new process ...
	cl. 18	+	cl. 44	... whithin lists...
	cl. 19	+	cl. 45	... said data and said program modules are copied into said first process
7	cl. 22	+	cl. 48	said process may be transferred ...
	cl. 23	+	cl. 49	... a construct is formed at least some of the data and/or program modules and execturion states
	cl. 24	+	cl. 50	... a subset of the data, program modules and execution states ...
	cl. 25	+	cl. 51	... allows to run in the second hardware ...
	cl. 26	+	cl. 52	... is a memory storage device ...

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/SG 98/00102

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US A 5603031	11-02-1997	AU A1 72164/94	06-02-1995
		EP A2 634719	18-01-1995
		EP A3 634719	03-01-1996
		JP A2 7182174	21-07-1995
		JP T2 7509799	26-10-1995
		WO A1 9502219	19-01-1995
		US A 6016393	18-01-2000
WO A1 9735262	25-09-1997	none	

09/856513

PATENT COOPERATION TREATY

REC'D 10 SEP 2001
WIPO PCT

PCT

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

(PCT Article 36 and Rule 70)

Applicant's or agent's file reference FP1103	**FOR FURTHER ACTION** See Notification of Transmittal of International Preliminary Examination Report (Form PCT/IPEA/416)	
International application No. PCT/SG 98/00102	International filing date (*day/month/year*) 16 December 1998 (16.12.1998)	Priority Date (*day/month/year*)
International Patent Classification (IPC) or national classification and IPC IPC^7: G06F 9/46, G06F 9/54		RECEIVED OCT 31 2001
Applicant Kent Ridge Digital Labs et al.		Group 2100

1. This international preliminary examination report has been prepared by this International Preliminary Examination Authority and is transmitted to the applicant according to Article 36.
2. This REPORT consists of a total of 3 sheets, including this cover sheet.

   ☒ This report is also accompanied by ANNEXES, i.e., sheets of the description, claims and/or drawings which have been amended and are the basis for this report and/or sheets containing rectifications made before this Authority (see Rule 70.16 and Section 607 of the Administrative Instructions under the PCT).

   These annexes consist of a total of 5 sheets.
3. This report contains indications relating to the following items:

   - I. ☒ Basis of the opinion
   - II. ☐ Priority
   - III. ☐ Non-establishment of opinion with regard to novelty, inventive step and industrial applicability
   - IV. ☐ Lack of unity of invention
   - V. ☒ Reasoned statement under Rule 66.2(a)(ii) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement
   - VI. ☐ Certain documents cited
   - VII. ☐ Certain defects in the international application
   - VIII. ☐ Certain observations on the international application

Date of submission of the demand	Date of completion of this report
30 June 2000 (30.06.2000)	25 July 2001 (25.07.2001)
Name and mailing address of the IPEA/AT Austrian Patent Office Kohlmarkt 8-10 A-1014 Vienna Facsimile No. 1/53424/200	Authorized officer FASTENBAUER Telephone No. 1/53424/

Form PCT/IPEA/409 (cover sheet) (July 1998)

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.
PCT/SG 98/00102

I. Basis of the report

1. With regard to the **elements** of the international application:*

☐ the international application as originally filed

☒ the description:
pages 1-25, as originally filed
pages ______, filed with the demand
pages ______, filed with the letter of ______.

☒ the claims:
pages ______, as originally filed
pages ______, as amended (together with any statement) under Article 19
pages ______, filed with the demand
pages 1-5, filed with the letter of 16 July 2001 (16.07.2001).

☐ the drawings:
pages ______, as originally filed
pages ______, filed with the demand
pages ______, filed with the letter of ______.

☐ the sequence listing part of the description:
pages ______, as originally filed
pages ______, filed with the demand
pages ______, filed with the letter of ______.

2. With regard to the **language**, all the elements marked above were available or furnished to this Authority in the language in which the international application was filed, unless otherwise indicated under this item.
These elements were available or furnished to this Authority in the following language ______ which is:

☐ the language of a translation furnished for the purposes of international search (under Rule 23.1(b)).

☐ the language of publication of the international application (under Rule 48.3(b)).

☐ the language of the translation furnished for the purposes of international preliminary examination (under Rule 55.2 and/or 55.3).

3. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, the international preliminary examination was carried out on the basis of the sequence listing:

☐ contained in the international application in printed form.

☐ filed together with the international application in computer readable form.

☐ furnished subsequently to this Authority in written form.

☐ furnished subsequently to this Authority in computer readable form.

☐ The statement that the subsequently furnished written sequence listing does not go beyond the disclosure in the international application as filed has been furnished.

☐ The statement that the information recorded in computer readable form is identical to the written sequence listing has been furnished.

4. ☒ The amendments have resulted in the cancellation of:

☐ the description, pages ______.

☒ the claims, Nos. 28-52.

☐ the drawings, sheets/fig ______.

5. ☐ This report has been established as if (some of) the amendments had not been made, since they have been considered to go beyond the disclosure as filed, as indicated in the Supplemental Box (Rule 70.2(c)).**

* *Replacement sheets which have been furnished to the receiving Office in response to an invitation under Article 14 are referred to in this report as „originally filed" and are not annexed to this report since they do not contain amendments (Rules 70.16 and 70.17).*

** *Any replacement sheet containing such amendments must be referred to under item 1 and annexed to this report.*

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.
PCT/SG 98/00102

V. Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. Statement

Novelty (N)	Claims	1-27	YES
	Claims		NO
Inventive step (IS)	Claims	1-27	YES
	Claims		NO
Industrial applicability (IA)	Claims	1-27	YES
	Claims		NO

Citations and explanations (Rule 70.7)

In view of the amended claims 1-27, the documents cited in the search report are no longer of particuar relevance. The new claims meet the criteria of Novelty, Inventive step and Industrial applicability.

CLAIMS

1. A computing environment having a plurality of computer components, each of the computer components being suitable for running a process which comprises a plurality of objects each comprising data and program modules operating on the data, and which has an execution state at any time, the computing environment supporting:

transmittal of a process running on any first one of the computer components to any second one of the computer components to resume running on the second computer component, and

evolution of the process by selective deletion of objects from within the process and/or the selective addition of objects into the process and/or the selective replacement of objects within the process by new objects, thereby changing the functionality of the process.

2. A computing environment as claimed in claim 1 wherein a construct is formed comprising data and/or program modules and execution state of a first process, and wherein said evolutionary operations are performed by functions operating on a said construct.

3. A computing environment as claimed in claim 2 wherein said construct is formed by a construct operation that suspends all active threads of said first process and creates a new process comprising at least some of the data and/or program modules and execution state of said first process, and stores said new process in a data area of said first process.

4. A computing environment as claimed in claim 3 wherein said construct comprises only data, program modules and execution state falling within lists that are passed to said construct operation.

5. A computing environment as claimed in any of claims 2 to 4 wherein said construct is provided with an authorising signature.

6. A computing environment as claimed in any preceding claim wherein said evolutionary operations include the selective deletion of objects from within said process.

7. A computing environment as claimed in any preceding claim wherein said evolutionary operations include the selective loading or reloading of objects into said process.

8. A computing environment as claimed in claim 1 wherein a said evolutionary operation includes the incorporation into a first process of new objects from a second process.

9. A computing environment as claimed in claim 8 wherein a construct is formed comprising at least some of the data and/or program modules and execution state from said second process, and said construct is transferred to said first process.

10. A computing environment as claimed in claim 9 wherein said construct is formed by a construct operation that suspends all active threads of said second process and creates a new process comprising a subset of the data, program modules and execution state of said second process, and stores said new process in a data area of said second process.

11. A computing environment as claimed in claim 10 wherein said construct comprises only data, program modules and execution state falling within lists that are passed to said construct operation.

12. A computing environment as claimed in any of claims 9 to 11 wherein said construct is provided with an authorising signature.

13. A computing environment as claimed in any of claims 9 to 12 wherein after said construct is transferred the second process stored within said construct is caused to be activated within said first process.

14. A computing environment as claimed in any of claims 9 to 12 wherein after said construct is transferred the first process is suspended and the second process stored within said construct is activated, and when the second process is concluded the data and program modules of the second process are added to the first process and the first process is re-activated.

15. A computing environment as claimed in any of claims 9 to 12 wherein after said first process terminates at least some of the data and/or program modules from said first process are added to the second process stored in said construct and said second process is then activated.

16. A computing environment as claimed in claim 8 wherein a construct is formed comprising at least some of the data and/or program modules from said second process, and said construct is transferred to said first process.

17. A computing environment as claimed in claim 16 wherein said construct is formed by a construct operation that suspends all active threads of said second process and creates a new process comprising at least some of the data and/or program modules of said second process, and stores said new process in a data area of said second process.

18. A computing environment as claimed in claim 17 wherein said construct comprises only a subset of data and program modules falling within lists that are passed to said construct operation.

PCT/ SG98/00102
16 Juli 2001

19. A computing environment as claimed in any of claims 16 to 18 wherein said data and said program modules from said second process are copied into said first process.

20. A computing environment as claimed in any of claims 8 to 19 wherein in the event of a conflict between data and/or program modules of said first process and data and/or program modules of said second process, the data and/or program modules of said first process will override the data and/or program modules of said second process.

21. A computing environment as claimed in any of claims 8 to 19 wherein in the event of a conflict between data and/or program modules of said first process and data and/or program modules of said second process, the data and/or program modules of said second process will override the data and/or program modules of said first process.

22. A computing environment as claimed in claim 1 wherein a said process may be transferred between different first and second hardware components of said computing environment.

23. A computing environment as claimed in claim 22 wherein a construct is formed comprising at least some of the data and/or program modules and execution state of said process, and said construct is transferred.

24. A computing environment as claimed in claim 22 wherein said construct comprises a subset of the data, program modules and execution state of said process.

25. A computing environment as claimed in any of claims 22 to 24 wherein a said process is subject to an evolutionary operation that allows the process to run in the second hardware component.

26. A computing environment as claimed in any of claims 22 to 24 wherein said second hardware component is a memory storage device.

27. A method of operating a computer process within a computing environment defined by a plurality of computer components, the process comprising a plurality of objects which each comprise data and program modules operating on the data, the process running on a first of the computer components and having an execution state at any time, the method comprising:

transmitting the process to a second of the computer components, the process resuming operation on the second computer component, and

modifying the process by selective deletion of objects from within the process and/or the selective addition of objects into the process and/or the selective replacement of objects within the process by new objects thereby changing the functionality of the process.